PROCEEDINGS

55. IWK

Internationales Wissenschaftliches Kolloquium
International Scientific Colloquium

13 - 17 September 2010

Crossing Borders within the ABC

Automation,

Biomedical Engineering and

Computer Science

Faculty of
Computer Science and Automation

www.tu-ilmenau.de

TECHNISCHE UNIVERSITÄT
ILMENAU

Impressum
Published by

Publisher:	Rector of the Ilmenau University of Technology Univ.-Prof. Dr. rer. nat. habil. Dr. h. c. Prof. h. c. Peter Scharff
Editor:	Marketing Department (Phone: +49 3677 69-2520) Andrea Schneider (conferences@tu-ilmenau.de) Faculty of Computer Science and Automation (Phone: +49 3677 69-2860) Univ.-Prof. Dr.-Ing. habil. Jens Haueisen
Editorial Deadline:	20. August 2010
Implementation:	Ilmenau University of Technology Felix Böckelmann Philipp Schmidt

USB-Flash-Version.

Publishing House:	Verlag ISLE, Betriebsstätte des ISLE e.V. Werner-von-Siemens-Str. 16 98693 Ilmenau
Production:	CDA Datenträger Albrechts GmbH, 98529 Suhl/Albrechts
Order trough:	Marketing Department (+49 3677 69-2520) Andrea Schneider (conferences@tu-ilmenau.de)

ISBN: 978-3-938843-53-6 (USB-Flash Version)

Online-Version:

Publisher:	Universitätsbibliothek Ilmenau ilmedia Postfach 10 05 65 98684 Ilmenau

Home / Index:
http://www.db-thueringen.de/servlets/DocumentServlet?id=16739

GRAPH BASED APPROACH TO TEST BENCH CONSTRUCTING FOR DATAPATH

Andrei Karatkevich[*], *Samary Baranov***

*Institute of Computer Science and Electronics, University of Zielona Góra,
ul. Podgórna 50, 65-246 Zielona Góra, Poland
** School of Engineering, Bar Ilan University, Ramat Gan 52900, Israel

ABSTRACT

Testing a data path in a digital system such as a microprocessor requires checking every possible way of sending data between the functional units. This paper considers a task of generating a test bench for given Data Path, which covers every way of data sending and number of simulations of such ways is minimized. We present a method in which a data path is modeled by an oriented graph. The task of optimal test bench generation is formulated as task of covering all arcs by the paths from input to output nodes of the graph with minimal total number of the arcs. Merging input and output nodes reduces the task to the Chinese Postman Problem, where the set of paths can be obtained from the circuit being a solution of the CPP. The proposed method is illustrated by a case study of testing a data path of a simple processor.

Index Terms – High level synthesis, Data path, Transaction Level Model, test bench, simulation, verification, graphs, Algorithmic state machines.

1. INTRODUCTION

A digital system, especially a processing unit, usually can be considered as a composition of Data Path and Control Unit. Design testing is one of the necessary steps of system design.

If a design description in a hardware design language as VHDL is obtained, a test bench also specified in VHDL should be used to perform its simulation. It is a special code with the structure presented in Fig. 1. The test bench contains a design as a component (DUT –-Design Under Test), Stimuli –- the set of input vectors for the design and expected values which should be at the design output for each input vector. An EDA (Electronic Design Automation) simulation tool sends stimuli, one after another, to the DUT, and compares the outputs from the DUT with expected outputs. If they are different, then the designer receives information about a mistake [1].

Because of different structure and functions of Data Path and Control Unit it is reasonable to test them separately, constructing special test bench for each of them. For testing a data path it is necessary to check every direct connection between the functional units at least once. But we don't have possibilities to write something into *any* component of Data Path, because some of them get data only from other components. But there are "input" units, which get data from outside; and we can write data into them for test purposes. Suppose that a data path is represented as an oriented graph, where nodes correspond to the components of operational units, and arcs correspond to direct data sending between the units (a connection graph). Then the task of checking every connection is reduced to the task of covering all arcs of the connection graph by the paths from its source nodes to target nodes. Of course, in general case some of the connections have to be covered more than once.

Note that if we are going to test a kind of a process unit, we have to check every (micro)instruction rather than just every connection. That means that if the same connection is realized, say, by two microinstructions *Y1* and *Y2*, then two arcs in the graph should correspond to this connection, one labeled with *Y1*, another with *Y2*. Then the data path will be represented by a multigraph.

Usually such test benches are designed manually. The task of their automatic generation, which arises for complex designs, requires developing of appropriate algorithm, which is able to find a necessary graph covering with the minimized duplication of arcs.

2. IDEA OF PROPOSED METHOD

How to cover an oriented graph by the paths from its sources to targets with a minimal total number of arcs? That is the graph-theoretic form of the considered problem.

Suppose that all sources and targets of the considered connection graph G_1 are merged into one node v_0. If the obtained graph G_2 is Eulerian, then the Eulerian circuit provides an optimal solution: if we "cut" it into the shortest paths starting and ending in v_0, we obtain a set of paths from the sources to targets of the initial graph such that they cover every edge exactly once. But, of course, graph G_2 usually is not Eulerian. But, as far as every unit of Data Path is reachable from some input, and from every unit an output is reachable, G_2 is strongly connected by construction, and there exists a circuit visiting all its arcs.

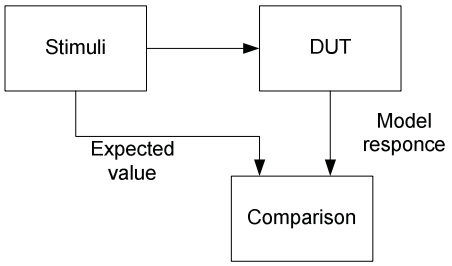

Figure 1 Test bench structure

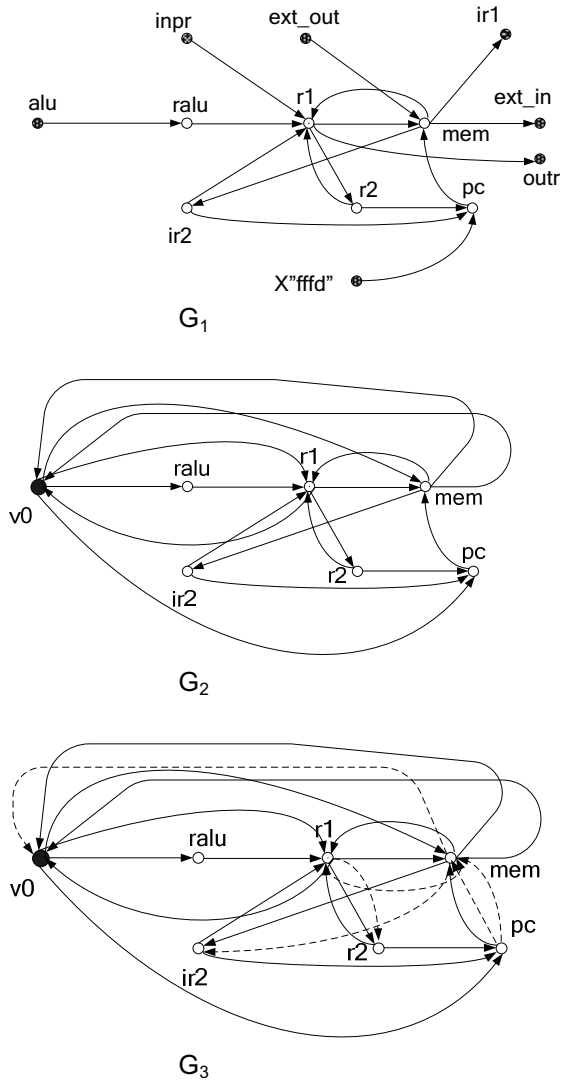

Figure 2 Illustration of the idea of the method

Then we have to find for graph G_2 the shortest of such paths. This is one of the variants of the route inspection problem, also known as the Chinese Postman Problem [5]. It can be effectively solved by duplicating some arcs in given graph in such a way, that new paths are added, each starting in a node which in G_2 has indegree greater than outdegree (let us denote set of such nodes of G_2 as I) and ending in a node which has outdegree greater than indegree (we denote set of such nodes as O). Every node of the resulting graph (G_3) must have equal indegree and outdegree, which guarantees that G_3 is Eulerian.

We have to introduce such paths with minimal sum length to obtain optimal solution. To do that, it is necessary to find the shortest paths from every node belonging to I to every node belonging to O (all paths that can be used for making the graph Eulerian). As far as our graph is not weighted, and a path length is measured by number of visited arcs, for obtaining such paths it is enough to perform simple breadth-first search [4]. BFS is linear; its applying to every node has quadratic complexity.

Then, to select the subset of the obtained paths with minimal length, which transforms graph G_2 into Eulerian one, it is necessary to solve an assignment problem [3]. It can be solved in polynomial time by the Hungarian algorithm [3][6]. An example of graphs G_1 (corresponding to a simple data path; microinstructions marking the arcs are not shown), G_2 and G_3 is shown in Fig. 2. Newly introduced arcs in G_3 are dotted.

When the Eulerian graph G_3 is obtained, Eulerian circuit for it can be constructed in linear time by using Fleury's algorithm [9]. From this circuit the shortest circuit visiting all arcs in G_3 can be directly obtained, and from it – the set of paths we are looking for, which cover all arcs in G_1.

In general case we cannot just construct a test bench from the concatenation of the obtained paths, because the same microinstruction (if it consists of several microoperations) may label more than one arc. We should avoid the situations in which a microoperation reads data from a unit, to which no data was written yet. So when constructing a test bench from the obtained paths, we should postpone using a microoperation in such situation until the unit will be initialized by another microoperation.

The proposed method deals only with constructing of "procedures to do" part of test bench – a sequence of microinstructions. Other test bench elements are not considered here. The data path is considered as a Transaction Level Model (TLM) [7], where the emphasis is more on the connections between the units and less on their actual implementation.

3. ALGORITHM OF TEST BENCH CONSTRUCTION

1. Create an oriented graph G_1, where a node corresponds to a data path unit and an arc corresponds to possible data transfer between two units by means of one microoperation. Label every arc with the microinstruction containing corresponding microoperation.
2. If there is a simple path in G_1 such that a) every node it passes, except the first one and the last one, has $deg^- = deg^+ = 1$ and b) every arc belonging to

the path is labeled with a microinstruction containing only one microoperation, then replace every such path by single arc from its beginning to its end (removing all arcs and internal nodes of the path) and label this new arc by concatenation of the labels of the removed arcs.

3. Merge all source and target nodes (with $deg^- = 0$ or $deg^+ = 0$) into one node v_0, obtaining graph G_2.

4. If G_2 is an Eulerian graph, then $G_3 := G_2$ and go to 8.

5. Apply BFS to every node v such that $deg^-(v) - deg^+(v) > 0$ (i.e., for every node belonging to I) to obtain all shortest paths leading from v to every node belonging to O.

6. Solve the assignment problem to select the paths with minimal total number of arcs for making the graph Eulerian. To do that, apply the Hungarian method.

7. Add the paths selected at step 6 to the graph by duplicating the arcs visited by every such path. Obtain in this way graph G_3.

8. Construct an Eulerian circuit for graph G_3 by using Fleury's algorithm. The same circuit is a solution of the Chinese Postman Problem for G_2.

9. Divide the obtained circuit into the set of shortest cycles starting and ending in v_0. Every such cycle in G_2 corresponds to a path from a source to a target in G_1, and all these paths together cover all the arcs in G_1.

10. Construct the sequence of microinstructions for the test bench in the following way:

a. Select the first arc not marked as "covered" in any path obtained in step 9 (initially none of them are marked); let it be labeled with *Yi*. If there are no uncovered arcs in another path labeled with the same microinstruction, add *Yi* to the sequence of microinstructions and mark the arc as "covered". If there are such uncovered arcs, and every first uncovered arc on a path marked with *Yi* has preceding covered arc, then add *Yi* to the sequence and mark first uncovered arc in every path, labeled with *Yi*, as "covered". Else select the first uncovered mark in another path and do the same.

b. Repeat 10.a while there are uncovered arcs in any path obtained in step 9.

Note that if there are uncovered arcs, but no microinstruction can be added in step 10a, then something is evidently wrong with the project. Suppose that there are two paths covering the connection graph labeled *Y1Y2* and *Y2Y1*, correspondingly. Then our algorithm in item 10.a will be "deadlocked", but that also would mean that, independently of the order of microinstructions, some data will be read from a unit to which no data was written before.

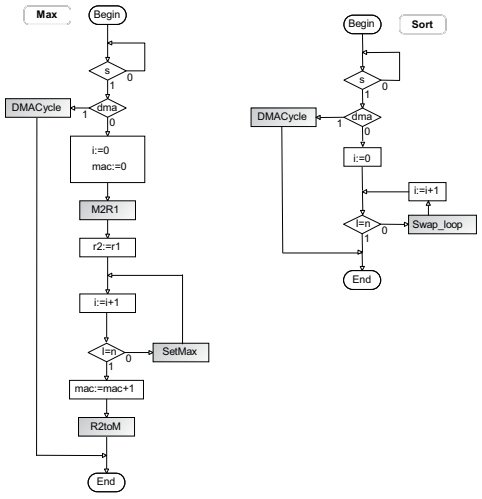

Figure 3 ASMs for the example

4. AN EXAMPLE

4.1. Data Path Design

We use the experimental EDA tool Abelite, which implements special algorithms for Data Path and Control Unit design and a very fast optimizing synthesis of FSM and combinational circuits [1][2]. At the considered stage Abelite uses a sequence of programs for automatic design of Data Path. In our design we use the common model in which any digital system is regarded as a composition of *Control Unit* and *Data Path*. Control unit produces a sequence of control signals that force implementation of microoperations in Data path.

One of the main concepts of Abelite design methodology is the construction of "*naked data path*", which doesn't contain any cloud circuits, only standard regular units with their inputs and outputs. Such units can be predesigned or even taken from the libraries. The Data path design contains several steps. Two first steps are:

1. *Construction of Connection Graph*. Abelite constructs this graph and then the list of parallel microoperations from the functional Algorithmic State Machine (ASM), obtained at the step of algorithmic design, separately for each length of transfers. Such a graph contains the list of sources and targets for each component of an operational unit and some metrics that are used in the optimization of Data Path.

2. *Construction of the optimized list of parallel microoperations* from the functional ASM. Such a list contains microoperations which should be implemented in parallel. If several microoperations are in one microinstruction, they are implemented concurrently (at the same clock).

The following steps deal with the implementation details and are not important here. They are explained in [1][2].

4.2. A Simple Processor

As an example we use a design of a very simple processor implementing two operations – the bubble sort and the search of the maximal element (Fig. 3 demonstrates high level of ASMs for these operations, the dark operator vertices are generalized operators). It is an improved version of the design described in [2]. List of its microinstructions is presented in Tab. 1.

Let us apply the algorithm described in section 3. The connection graph for the given example is presented in Fig. 4.

Graph G_2 (item 3 of the algorithm) is shown in Fig. 5. Here the dark nodes from G_1 (*alu16*, *ext_adr*, *m_adr*, *ext_out*, *ext_in*, *0*, *comp16_in1*, *comp16_in2*, *n*, *alu16_in1*) are merged, and node v_0 correspond them in G_2. This node looks untypically in Fig. 5, because otherwise it would be difficult to show a node with so many incoming and outgoing arcs.

Table 1. Functional microinstructions

Micro instr	Functional microoperations	
Y1	y16	i:=0
	y25	mac:=0
Y2	y28	mac:=mac+1
Y3	y31	r2:=r1
Y4	y17	i:=i+1
Y5	y16	i:=0
Y6	y20	m[m_adr]:=ext_out
	y23	m_adr:=ext_adr
Y7	y15	ext_in:=m[m_adr]
	y23	m_adr:=ext_adr
Y8	y24	m_adr:=mac
	y30	r1:=m[m_adr]
Y9	y26	mac:=i
Y10	y22	m[m_adr]:=r2
	y24	m_adr:=mac
Y11	y18	j:=0
Y12	y19	j:=j+1
Y13	y28	mac:=mac+1
	y31	r2:=r1
Y14	y29	mac:=mac-1
Y15	y27	mac:=j
Y16	y21	m[m_adr]:=r1
	y24	m_adr:=mac
Y17	y5	comp16_ctr:="011"
	y8	comp16_in1:=r1
	y12	comp16_in2:=r2
	y14	compFlag:=comp16
Y18	y4	comp16_ctr:="001"
	y7	comp16_in1:=j
	y13	comp16_in2:=tempReg16
	y14	compFlag:=comp16
Y19	y1	alu16_ctr:="010"
	y2	alu16_in1:=n
	y3	alu16_in2:=1
	y32	tempReg16:=alu16
Y20	y5	comp16_ctr:="011"
	y9	comp16_in1:=r2
	y11	comp16_in2:=r1
	y14	compFlag:=comp16
Y21	y4	comp16_ctr:="001"
	y6	comp16_in1:=i
	y10	comp16_in2:=n
	y14	compFlag:=comp16

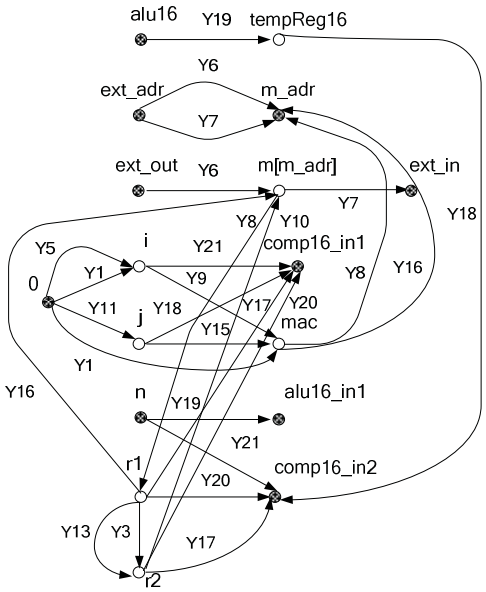

Figure 4 Connection graph (G_1)

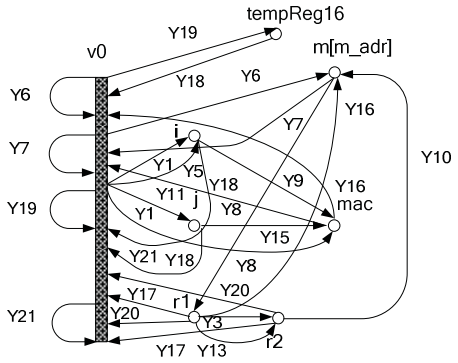

Figure 5 Graph G_2

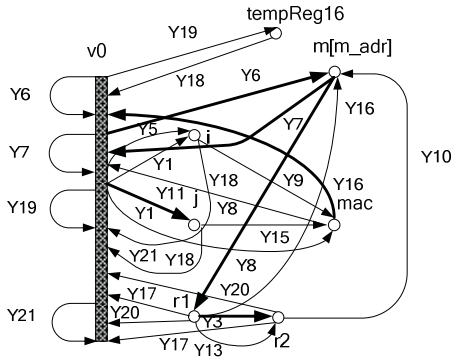

Figure 6 Graph G_2 with bolded arcs being candidates for duplicating

4.3. Test Bench for Data Path

It is easy to see that graph G_2 is not Eulerian. It has several nodes with non-zero difference between indegree and outdegree (deg^- - deg^+). These nodes and corresponding differences are:

v_0	4;	*r1*	-4;
m[m_adr]	1;	*r2*	-1;
mac	1;	*j*	-1.

New paths, needed to make the graph Eulerian, will lead from v_0, *m[m_adr]* and *mac* to *j*, *r1* and *r2*. To find shortest of such paths, we apply BFS to the first 3 of mentioned nodes. In Fig. 6 the arcs belonging to the paths obtained in such way are bolded. Only bolded arcs will be duplicated (but we don't know yet, whether all of them will be duplicated and how many times).

Lengths of the shortest paths obtained by means of DFS (measured in the number of arcs) are shown in Table 2.

Table 2. Lengths of the shortest paths obtained in the item 5 of the algorithm

	j	r1	r2
v_0	1	2	3
m[m_adr]	2	1	2
mac	2	3	4

To make the graph Eulerian, we have to introduce new paths such that 4 of them start in v_0, 1 in *m[m_adr]*, and 1 in *mac*; 4 of them end in *r1*, 1 ends in *r2*, and 1 in *j*. We have to take it into account to formulate conditions of the assignment problem. So in our matrix of the costs we will have 4 rows for v_0 (named as v_0^a - v_0^d) and 4 columns for *r1* ($r1^a$ - $r1^d$. The matrix of the costs is shown in Table 3.

Table 3. Matrix of the costs for the assignment problem

	j	$r1^a$	$r1^b$	$r1^c$	$r1^d$	r2
v_0^a	1	2	2	2	2	3
v_0^b	1	2	2	2	2	3
v_0^c	1	2	2	2	2	3
v_0^d	1	2	2	2	2	3
mac	2	3	3	3	3	4
m[m_adr]	2	1	1	1	1	2

Now we apply the Hungarian algorithm. The problem written in the form of a matrix is given below.

$$\begin{bmatrix} 1 & 2 & 2 & 2 & 2 & 3 \\ 1 & 2 & 2 & 2 & 2 & 3 \\ 1 & 2 & 2 & 2 & 2 & 3 \\ 1 & 2 & 2 & 2 & 2 & 3 \\ 2 & 3 & 3 & 3 & 3 & 4 \\ 2 & 1 & 1 & 1 & 1 & 2 \end{bmatrix}$$

Steps 1 and 2 of the Hungarian algorithm (the lowest element is subtracted from every element in every row, then in every column):

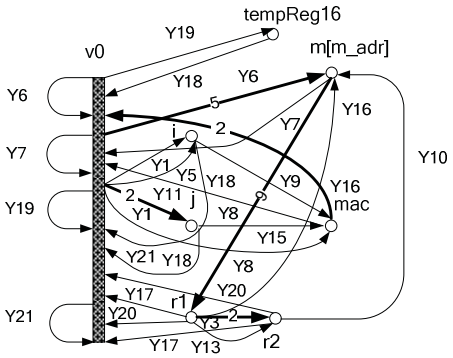

Figure 7 Graph G_3

$$\begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Step 3 and step 4 of the Hungarian algorithm (for details see [3]) lead to the following matrix:

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

That means that any assignment avoiding assigning *m[m_adr]* to j is minimal (has cost 12). So we can, for example, make assignment according to the main diagonal of the matrix. Then we duplicate the following paths: from v_0 to *j* and 3 times to *r1*, from *mac* to *r1*, and from *m[m_adr]* to *r2*. 12 arcs will be added.

It is illustrated by Fig. 7. To make it more readable, we do not draw additional arcs, writing number of arcs to which now corresponds every bolded arc instead. For example, arc *m[m_adr]*→*r1* appears 5 times in the additional paths (in all of them except path from v_0 to *j*), so there are 6 exemplars of this arc in the graph; arc *m[m_adr]*→v_0, however bolded in Fig. 6, is not duplicated. It is easy to see, that now every node has equal indegree and outdegree, so the graph is Eulerian now.

Eulerian path in graph G_3 *and* shortest cycle visiting every arc in G_2 is:

$$v_0 \xrightarrow{Y6} v_0 \xrightarrow{Y7} v_0 \xrightarrow{Y19} v_0 \xrightarrow{Y21} v_0 \xrightarrow{Y19} tempReg16 \xrightarrow{Y18} v_0 \xrightarrow{Y6} m[m\_adr] \xrightarrow{Y7} v_0 \xrightarrow{Y6} m[m\_adr] \xrightarrow{Y8} r1 \xrightarrow{Y17} v_0 \xrightarrow{Y6} m[m\_adr] \xrightarrow{Y8} r1 \xrightarrow{Y20} v_0 \xrightarrow{Y6} m[m\_adr] \xrightarrow{Y8} r1 \xrightarrow{Y3} r2 \xrightarrow{Y20} v_0 \xrightarrow{Y6} m[m\_adr] \xrightarrow{Y8} r1 \xrightarrow{Y3} r2 \xrightarrow{Y10} m[m\_adr] \xrightarrow{Y8} r1 \xrightarrow{Y16} m[m\_adr] \xrightarrow{Y8} r1 \xrightarrow{Y13} r2 \xrightarrow{Y17} v_0 \xrightarrow{Y1} i \xrightarrow{Y21} v_0 \xrightarrow{Y5} i \xrightarrow{Y9} mac \xrightarrow{Y16} v_0 \xrightarrow{Y11} j \xrightarrow{Y15} mac \xrightarrow{Y16} v_0 \xrightarrow{Y11} j \xrightarrow{Y18} v_0 \xrightarrow{Y1} mac \xrightarrow{Y8} v_0.$$

The paths for the connection graph, obtained from this cycle in item 9 of the algorithm, are shown below.

$$ext\_adr \xrightarrow{Y6} m\_adr$$
$$ext\_adr \xrightarrow{Y7} m\_adr$$
$$n \xrightarrow{Y19} alu16\_in1$$
$$n \xrightarrow{Y21} comp16\_in2$$
$$alu16 \xrightarrow{Y19} tempReg16 \xrightarrow{Y18} comp16\_in2$$
$$ext\_out \xrightarrow{Y6} m[m\_adr] \xrightarrow{Y7} ext\_in$$
$$ext\_out \xrightarrow{Y6} m[m\_adr] \xrightarrow{Y8} r1 \xrightarrow{Y17} comp16\_in1$$
$$ext\_out \xrightarrow{Y6} m[m\_adr] \xrightarrow{Y8} r1 \xrightarrow{Y20} comp16\_in2$$
$$ext\_out \xrightarrow{Y6} m[m\_adr] \xrightarrow{Y8} r1 \xrightarrow{Y3} r2 \xrightarrow{Y20} comp16\_in1$$
$$ext\_out \xrightarrow{Y6} m[m\_adr] \xrightarrow{Y8} r1 \xrightarrow{Y3} r2 \xrightarrow{Y10} m[m\_adr] \xrightarrow{Y8} r1 \xrightarrow{Y16} m[m\_adr] \xrightarrow{Y8} r1 \xrightarrow{Y13} r2 \xrightarrow{Y17} comp16\_in2$$
$$0 \xrightarrow{Y1} i \xrightarrow{Y21} comp16\_in1$$
$$0 \xrightarrow{Y5} i \xrightarrow{Y9} mac \xrightarrow{Y16} m\_adr$$
$$0 \xrightarrow{Y11} j \xrightarrow{Y15} mac \xrightarrow{Y16} m\_adr$$
$$0 \xrightarrow{Y11} j \xrightarrow{Y18} comp16\_in1$$
$$0 \xrightarrow{Y1} mac \xrightarrow{Y8} m\_adr.$$

Applying item 10 of the algorithm results in the following test sequence of the microinstructions:

Y6 Y7 Y19 Y1 Y21 Y8 Y3 Y20 Y10 Y8 Y5 Y9 Y11 Y18 Y15 Y16 Y8 Y13 Y17.

5. CONCLUSION AND FURTHER WORK

We proposed a new approach to formal construction of test bench for a data path. Sequence of microinstructions for data path is generated by constructing of shortest covering of the connection graph by paths from its sources to targets. Formalization of data path construction with minimization of test length is useful for big designs, especially taking into account, that the method we propose is exact and quick. Its slowest part is the Hungarian algorithm, which can be implemented with time complicity $O(n^3)$ [6].

The task of test bench construction for a system like microprocessor is complicated by the fact that the test sequence should consist of microinstructions, and a microinstruction may consist of several microoperations, hence one microinstruction may transfer data between more than two data path units. We solve this problem in item 10 of the proposed algorithm, which does not look elegant. It would be interesting to try another approach: modeling data path by a Petri net, where a transition corresponds to a microinstruction (which allows to model data transfer between several units), and looking for a firing sequence covering all transitions, probably using T-invariants [8]. It will be a topic of our future work.

6. REFERENCES

[1] Baranov, S., Logic and System Design of Digital Systems, TGU, Tallinn, 2008.

[2] Baranov, S., "ASMs in High Level Synthesis of EDA tool Abelite", Preprints of the 4th IFAC Workshop on Discrete-Event System Design, IFAC, Gandia Beach, pp. 195-200, 2009 (to appear online in the IFAC-PapersOnLine.net website).

[3] Burkard, R., M. Dell'Amico, and S. Martello, Assignment Problems, SIAM, Philadelphia, 2009.

[4] Cormen, T.H., Ch.E. Leiserson, R.L. Rivest, and C. Stein, Introduction to Algorithms (2nd ed.), MIT Press and McGraw-Hill, Massachusetts, 2001.

[5] J. Edmonds and E.L. Johnson, "Matching Euler tours and the Chinese postman problem," Mathematical Programming, pp. 88-124, No. 5, 1973.

[6] A. Frank, On Kuhn's Hungarian Method – A tribute from Hungary, Technical report, Egrerváry Research Group, Budapest, Hungary, 2004.

[7] Grötker, T., S. Liao, G. Martin, and S. Swan, System Design with SystemC, Chapter 8, Kluwer Academic Publisher, Boston, 2002.

[8] T. Murata, "Petri nets: Properties, Analysis and Application", Proceedings of the IEEE, pp. 541-580,

[9] Skiena, S.S. Implementing Discrete Mathematics: Combinatorics and Graph Theory with Mathematica, Addison-Wesley, Reading, MA, 1990.